

Efficient Sum-Based Hierarchical Smoothing Under ℓ_1 -Norm

Siavosh Benabbas*
siavosh@cs.toronto.edu

Hyun Chul Lee†
chul.lee@thoora.com

Joel Oren*‡
oren@cs.toronto.edu

Yuli Ye*‡
y3ye@cs.toronto.edu

August 9, 2011

Abstract

We introduce a new regression problem which we call the *Sum-Based Hierarchical Smoothing* problem. Given a directed acyclic graph and a non-negative value, called *target value*, for each vertex in the graph, we wish to find non-negative values for the vertices satisfying a certain constraint while minimizing the distance of these assigned values and the target values in the ℓ_p -norm. The constraint is that the value assigned to each vertex should be no less than the sum of the values assigned to its children. We motivate this problem with applications in information retrieval and web mining. While our problem can be solved in polynomial time using linear programming, given the input size in these applications such a solution is too slow.

We mainly study the ℓ_1 -norm case restricting the underlying graphs to rooted trees. For this case we provide an efficient algorithm, running in $O(n^2)$ time. While the algorithm is purely combinatorial, its proof of correctness is an elegant use of linear programming duality. We also present a number of other positive and negatives results for different norms and certain other special cases.

We believe that our approach may be applicable to similar problems, where comparable hierarchical constraints are involved, e.g. considering the average of the values assigned to the children of each vertex. While similar in flavour to other smoothing problems like Isotonic Regression (see for example [Angelov et al. SODA'06]), our problem is arguably richer and theoretically more challenging.

*Department of Computer Science, University of Toronto

†Thoora Inc., Toronto, ON, Canada

‡This research was supported by the MITACS Accelerate program, Thoora Inc., and The University of Toronto, Department of Computer Science.

1 Introduction

The prevalence of popular web services like Amazon, Google, Netflix, and StumbleUpon has given rise to many interesting large-scale problems related to classification, recommendation, ranking, and collaborative filtering. In several recent studies (e.g. [KFB09, PG08, CKP07]), researchers have incorporated the underlying class hierarchies of the data-sets into the setting of recommendation systems. Moreover, Koren et al. [DKK11] recently demonstrated an application of hierarchical classifications of topics, i.e. *taxonomies*, in Collaborative Filtering settings, in particular, music recommendation. In these application scenarios, the taxonomies are abstracted as trees. Associated with the vertices are scalar target values, typically inferred through the use of various machine learning or information retrieval methods. For instance, given a hierarchy of topics and a search query, the target values could be the relevance measures of the topics to the search query.

When a taxonomy is used, one would usually like to enforce particular constraints on the value assigned to the vertices to properly represent the hierarchical relationship among them. Typically, the relevant machine learning approaches are ill-equipped to handle these requirements. Often, these constraints state that the value of each vertex should be at least some function of the value of its direct children in the taxonomy (e.g. [PG08, CKP07]).

Going back to the previous example of topics and search query, imagine that the taxonomy contains the topics *sports*, *baseball*, *football*, and *basketball* with the first topic being the parent of the other three and that the search query is “ESPN”. One would like to find the relevance of this query to every topic in the taxonomy. A reasonable requirement of these relevance values would be that the relevance of “ESPN” to *sports* would be no less than the sum of its relevance to *baseball*, *football*, and *basketball*. One way to solve this problem would be to directly impose such a constraint on the learning algorithm that infers the relevance values using regularization; i.e. adding an additional term in the objective function of that algorithm penalizing any violation of the constraint. However, this approach has two problems. First, it “softens” our requirements; i.e. it allows for possible violations, to some limited extent. Moreover, it can dramatically deteriorate the running time of the process of learning or restrict our choice of the learning algorithm.

Instead, we take the following, widely used, two-step approach. Given a search query s , we first infer each of the relevance scores of each of the topics, disregarding the hierarchy constraints. Then, we *smoothen* the inferred relevance scores by modifying them so as to uphold the above sum constraint. We would want the change of the relevance scores in the second step to be as small as possible. As the relevance scores are scalar values, we can represent both the original and final relevance scores as two vectors with non-negative values, and measure their difference in a suitable norm (e.g. the ℓ_1 , ℓ_2 or the ℓ_∞ norms). The subject of this paper is how to perform the second step.

We formulate this problem which we call the *Sum-Based Hierarchical Smoothing* problem (SBHSP) as follows. Given a rooted tree (or in general a directed acyclic graph) $G = (V, E)$ and a vector of original vertex values (called *target values*) $\mathbf{a} = (a_{v_1}, a_{v_2}, \dots, a_{v_n})$ the objective is to find a vector of new vertex values (called *assigned values*) $\mathbf{x} = (x_{v_1}, x_{v_2}, \dots, x_{v_n})$ with the following properties. (i) for any node w with incoming edges $(u_1, w), \dots, (u_k, w)$ we have $x_{u_1} + \dots + x_{u_k} \leq x_w$. (ii) $\|\mathbf{a} - \mathbf{x}\|_p$ is minimized. Different values of p result in different variants of the problem. We mainly study the problem for $p = 1$ and $p = \infty$ but the case of $p = 2$ is also interesting. It is not hard to see that for $p = 1$ the problem can be solved in polynomial time using linear programming (see inequalities (3a)-(3d)) and for $p > 1$ it can be solved by using a suitable separation oracle and the Ellipsoid method. However given the typical size of taxonomies these solutions are too slow.

We note that this problem seems to be more complex than other previously considered similar problems as the assigned value of each vertex affects the possible values for any vertex it shares a parent with. In particular, to the best of our knowledge techniques used for similar problems are ineffective for it.

Contributions: Our main contribution is a purely combinatorial algorithm when the input is a rooted tree and $p = 1$ (i.e. the ℓ_1 norm) that runs in time $O(n^2)$. We note that the ℓ_1 norm was previously used as a good measure of difference in similar regression problems (e.g. see [AHKW06]). As many hierarchical structures in practice are trees, our algorithm can be used in many practical applications. Our second contribution is a linear time algorithm for the case $p = \infty$ which works for any directed acyclic graph. We also show an efficient FPTAS for optimizing the ℓ_1 norm for another class of DAGs (directed bilayer graphs.) Finally, we show that if one adds the extra condition that the assigned values should be *integral* the problem is hard to approximate (to within a polylogarithmic factor) for any ℓ_p norm for $1 \leq p < \infty$. Interestingly, given that our algorithm for the ℓ_1 norm on trees always outputs an integral solution this last result suggests that new ideas are needed to extend it to general DAGs.

Our algorithm for the ℓ_1 case has a rather simple structure. We assign values to the vertices of the tree in a bottom-up manner. For each vertex we first assign a valid (but possibly suboptimal) value and then use paths going down from that vertex to “push the excess” down the tree and improve the objective value. While the algorithm is purely combinatorial, its proof of correctness is an elegant use of linear programming duality. In particular, we use the complementary slackness condition to show that if the algorithm can no longer push the excess of a node down the tree the values assigned to its subtree must be optimal.

Organization: We present the relevant previous work in Section 2. In Section 3, we present a precise definition of the problem and some preliminaries. We present our first algorithm which is for the case of trees and ℓ_1 norm in Section 4 and prove its correctness. In Section 5 we show how this algorithm can be optimized to run in the promised $O(n^2)$ time. We conclude and propose several open problems in Section 6. We extend the algorithm to the case of *weighted* ℓ_1 norm in Appendix A. We present our algorithm for the case of ℓ_∞ in Appendix B. We leave our hardness of approximation result to Appendix C and our results for the case of bilayer graphs to Appendix D.

2 Previous Work

The main motivation of the current paper is the application of taxonomies in regression. A recent example, studied by Koren et al. [DKK11], is the application of topic hierarchies in the context of collaborative filtering. They provide a method of linking the data-set to a four level taxonomy, which helps them circumvent difficulties related to the size of the data-set.

Regression and smoothing problems have been studied extensively in recent years. Perhaps the most relevant problem to our setting is the *Isotonic regression* problem and its variants. There one wishes to find a closest fit to a given vector subject to a set of monotonicity constraints. More precisely, let $\mathbf{a} = \langle a_1, \dots, a_n \rangle$ be n target values, and let E be a set of m pairwise order constraints on these variables. The *Isotonic regression* problem is to find values $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ such that $x_i \geq x_j$ whenever $(i, j) \in E$ for which the distance between \mathbf{x} and \mathbf{a} is minimized. To put things in a language similar to ours, in isotonic regression the assigned value of each vertex should be bigger than the *maximum* of the assigned value of its children as opposed to the *sum* of those values in our problem.

Common choices of distance functions include the weighted ℓ_1 , ℓ_2 and ℓ_∞ norms. The Isotonic

regression problem for such weighted norms have been studied extensively. For some of the results for the ℓ_1 and ℓ_2 norms see [Sto08, AHKW06, BC90]. Stout also maintains a web site containing some of the fastest known Isotonic regression algorithms for different settings at [Sto].

The Isotonic regression problem belongs to a more general class of problems known as *order restricted statistical inference*. Order restricted statistical inference was first studied by Barlow et al [BBBB72]. The Isotonic regression problem became popular since it has many applications in testing [LB01, MAC01], modelling [MJDP⁺00, Ulm86], data smoothing [FT84, PG08] and other areas [RWD88] related to statistical and computational data analysis. It has been shown to be an important post-processing smoothing tool to impose desired hard constraints on the values that a learning algorithm has produced. Variations of Isotonic regression have been used for other applications like template learning [CKP07], ranking [DCZ⁺10, MSCZ10], and classification [KFB09].

3 Preliminaries

We now formally define the problem as follows. Given a tree (or DAG) $T = (V, E)$ rooted at node $r \in V$, and a vector $\mathbf{a} \in \mathbb{R}_{\geq 0}^n$ of the target values of the vertices. We wish to find the *closest* vector $\mathbf{x} \in \mathbb{R}_{\geq 0}^n$, in the ℓ_p -norm, so that for each node v , with children u_1, \dots, u_k , $x_v \geq x_{u_1} + x_{u_2} + \dots + x_{u_k}$.

While most of the paper addresses the case of $p = 1$, we also discuss the case of $p = \infty$ in Appendix B. Note that our hardness results apply to *all* $1 \leq p < \infty$.

For a vertex $u \in T$, we denote the set of nodes with edges to u the *children* of u or $C(u)$, similarly the parent of u is $A(u)$ (in the case where the underlying graph is a general DAG, $A(u)$ will be a set of nodes). Throughout the paper, we will make extensive use of various paths in the given tree. For this purpose, we let $P_{u \rightarrow v}$ denote the (unique) path from vertex u to vertex v in T . We denote the sub-tree rooted in vertex v by T_v . For a given sub-tree T_v , we define $\mathbf{a}|_{T_v}$ as the vector of target values corresponding to the nodes in T_v ; we similarly define $\mathbf{x}|_{T_v}$.

4 The Algorithmic Approach for ℓ_1

As an initial attempt, consider the following trivial feasible solution. For each leaf $\ell \in T$, set $x_\ell = a_\ell$. Then, for each internal node v set $x_v = \max(a_v, \sum_{u \in C(v)} x_u)$, by traversing the tree in post-order. However, it is not hard to see that this approach would be arbitrarily sub-optimal (see Figure 1.) Indeed, in some cases it is preferable to lower the existing x values of a given node's children, instead of raising the node's x value, as this might help the objective value on the nodes ancestors as well.

In order to optimize the objective function, our algorithm will proceed as follows. By traversing the tree T in post-order, it performs the following sequence of steps for every vertex v . x_v is initially set to the maximum of a_v and the sum of the x values of its children, which is clearly a feasible assignment for T_v . It then improves the assignments for T_v by sequentially decreasing the values of some vertices that are located on some path P from v to some other node in T_v . The adjustments are made so that the overall improvement in the objective function equals the improvement in $|a_v - x_v|$. We will refer to such paths as *push-paths*, and the improvements made on them as *push operations*. The algorithm is presented below as Algorithm 1. The procedure *Push-Path*(\mathbf{x}, P, ϵ) checks what is the improvement on the objective function value if we reduce the x value of all vertices in the path P by ϵ . This path will always start at the current vertex v .

For now we do not discuss how to find the push path or the exact value that we push down that path. This abstraction was made deliberately, so as to separate the correctness of the algorithm from its performance. In fact, we later show that the individual paths need not be enumerated separately.

Algorithm 1: Push-Improve

Input: Undirected tree $T = (V, E)$, with a vector of vertex weights $\mathbf{a} \in \mathbb{R}_+^n$

Output: A feasible vector of weights $\mathbf{x} \in \mathbb{R}_+^n$ for V

```
1 Let  $v_1, v_2, \dots, v_{n-1}, v_n$  be the vertices in  $T$  sorted in post-order.
2 for  $v \leftarrow 1$  to  $n$  do
3    $x_v = \max\{\sum_{u \in C(v_i)} x_u, a_v\}$ 
4   ImproveSubtree( $v$ )
5 end

6 ImproveSubtree(Vertex  $u$ )
7 while  $\exists$  path  $P$  from  $u$  down to a vertex  $v$ , and  $\epsilon > 0$  such that  $v$  is either a leaf or
    $x_v > \sum_{w \in C(v)} x_w$  and Push-Path ( $\mathbf{x}, P, \epsilon$ )  $= \epsilon$  do
8   Push-Path( $\mathbf{x}, p, \epsilon$ )
9 end

10 Push-Path(Assignment  $\mathbf{x}$ , Path  $P$ , Non-negative real-value  $\epsilon$ )
11 begin
12   Let  $v_1, \dots, v_k$  be the sequence of nodes on the  $P$  from top to bottom.
13    $old = \sum_{1 \leq i \leq k} |x_{v_i} - a_{v_i}|$ 
14    $x_{v_1} = x_{v_1} - \epsilon$ 
15   for  $i = 2$  to  $k$  do
16      $t = \sum_{u \in C(v_{i-1})} x_u - x_{v_{i-1}}$ 
17     if  $t > 0$  then
18        $x_{v_i} = x_{v_i} - t$ 
19     end
20   end
21    $new = \sum_{1 \leq i \leq k} |x_{v_i} - a_{v_i}|$ 
22   return  $old - new$ 
23 end
```

The following theorem states that the output of Algorithm 1 is optimal.

Theorem 4.1. *When Algorithm 1 terminates, the obtained vector \mathbf{x} is a feasible and optimal assignment for the given tree T .*

Our proof of Theorem 4.1 will proceed as follows. We begin by characterizing the necessary push-path improvement at each step of the while-loop. We then inductively argue that before and after each push operation, the value of the objective function for each sub-tree rooted in a child of the current node remains optimal. We conclude by using an LP duality argument in order to show that once no more push operations exist for the current vertex in the for-loop, T_v is assigned optimal x values.

The following lemma refers to the series of improvements performed on node v , and can be viewed as the set of invariants of the outer for-loop.

Lemma 1. *Let v be the current node, $P = (v = u_0, \dots, u_k)$ be a push-path such that for $1 \leq i \leq k$, $u_i \in C(u_{i-1})$. Then the following invariants hold throughout the execution of the inner while-loop:*

1. If, for $\epsilon > 0$, $Push - Path(\mathbf{x}, P, \epsilon) > 0$, then $Push - Path(\mathbf{x}, P, \epsilon) \leq \epsilon$. Furthermore, if for path P and $\epsilon > 0$, $Push - Path(\mathbf{x}, P, \epsilon) = \delta > 0$, then there exists $\epsilon' > 0$ such that $Push - Path(\mathbf{x}, P, \epsilon') = \epsilon'$.
2. If for path P and $\epsilon > 0$ $Push - Path(\mathbf{x}, P, \epsilon) = \epsilon$, then for each $u \in C(v)$, T_u is optimally set before and after running $Push - Path(\mathbf{x}, P, \epsilon)$.

Proof. First, notice that the above invariants clearly hold if the current node v is a leaf, as their initial x values are set to their a values, and will only be modified as a result of performing $Push - Path$ on their ancestors. Assume that the invariants hold for all nodes preceding v in the post-order, and suppose for contradiction that there exists some path $P = (v = u_0, \dots, u_m)$ and $\epsilon > 0$ such that $Push - Path(\mathbf{x}, P, \epsilon) > \epsilon$. The first part of the first invariant clearly holds since the sub-trees rooted in the children of v are assumed optimal. Hence, any ϵ -improvement on v cannot entail an additional improvement on the rest of the push-path.

We now consider the second invariant, while briefly deferring the proof of the second part of the first invariant. First, notice that for each $\ell \in C(v) - \{u_1\}$, the assignments to T_ℓ do not change. Let P be a modification-path, and $\epsilon > 0$ such that $Push - Path(\mathbf{x}, P, \epsilon) = \epsilon$. On the other hand, notice that x_v is reduced by exactly ϵ . This implies that $\|\mathbf{x}|_{T_{u_1}} - \mathbf{a}|_{T_{u_1}}\|$ remains unchanged, thereby remaining optimal.

We now turn to the remaining part of the first invariant. Consider a modification-path P and $\delta > 0$. By the first part of the invariant, $Push - Path(\mathbf{x}, P, \delta) \leq \delta$. If $Push - Path(\mathbf{x}, P, \delta) = \delta$, then the claim holds trivially. Hence, assume $Push - Path(\mathbf{x}, P, \delta) < \delta$.

We restrict ourselves to dealing with δ values in the range $(0, x_v - a_v]$. The following observation stems from the fact that during the push operation, x values along P only decrease.

Observation 1. For path P and $\epsilon > 0$, if $Push - Path(\mathbf{x}, P, \epsilon) > 0$

$$|\{j \in P : x_j > a_j\}| > |\{j \in P : x_j \leq a_j\}| \quad (1)$$

In fact, using the induction hypothesis, we can make Observation 1 even stronger:

Claim 1. For path P and $\epsilon > 0$, if $Push - Path(\mathbf{x}, P, \epsilon) > 0$

$$|\{j \in P : x_j > a_j\}| - |\{j \in P : x_j \leq a_j\}| = 1 \quad (2)$$

Claim 1 can be justified by noticing that otherwise, the sub-tree rooted in one of v 's children would be amenable to path-improvements, contradicting optimality. The invariant follows, as we could simply set ϵ to be the minimum (positive) amount that maintains the number of nodes along P with x values that are larger than their a values. \square

Lemma 1 implies that each push operation improves the value of the objective function for the current sub-tree, while maintaining the optimality of the sub-trees rooted in the children of v . However, in order to show that the local optimum obtained by the algorithm is the globally optimal feasible solution, we need to argue that as long as the current assignment is not optimal, there exists a feasible path-improvement with a corresponding $\epsilon > 0$ value. The following theorem, which constitutes the main technical part of this paper, formalizes this notion.

Theorem 4.2. Upon termination of the inner while-loop, the sub-tree rooted in vertex v is assigned optimal x values.

Proof. First, notice that the algorithm clearly maintains the feasibility of the solution throughout its execution. The following observation follows from the definition of the algorithm.

Observation 2. *During the execution of the algorithm, $x_v \geq a_v$. Furthermore, if $x_v = a_v$, the solution is trivially optimal.*

The proof of Theorem 4.2 will proceed as follows. We give the LP for the optimization problem, and its corresponding dual LP. We then construct a feasible solution for the dual LP that satisfies the complementary slackness conditions with respect to the solution of the algorithm. In order to construct a valid dual solution, we inductively bootstrap the dual solutions constructed for the nodes rooted sub-trees. From LP duality, we then conclude that the two solutions are optimal for the primal and dual problems. Recall that we inductively assume that the sub-trees rooted in the children of v are optimally adjusted.

It is not hard to write a linear program which formulates our problem. This program and its dual can be seen below. The variables d_i are introduced to avoid using absolute values in the objective function.

$$\begin{aligned}
\min \quad & \sum_{i \in T_v} d_i & \max \quad & \sum_{i \in T_v} a_i(\lambda_i - \lambda'_i) \\
\text{subject to} \quad & d_i + x_i \geq a_i & \text{subject to} \quad & \lambda_i + \lambda'_i = 1 \quad \forall i \in T_v \quad (4a) \\
& d_i - x_i \geq -a_i & & (\lambda_i - \lambda'_i) + \alpha_i - \alpha_{p(i)} \leq 0 \quad \forall i \in T_v \setminus \{v\} \quad (4b) \\
& x_i - \sum_{j \in C(i)} x_j \geq 0 & & (\lambda_v - \lambda'_v) + \alpha_v \leq 0 \quad (4c) \\
& x_i \geq 0 \quad \forall i \in T_v & & \lambda_i, \lambda'_i, \alpha_i \geq 0 \quad \forall i \in T_v \quad (4d)
\end{aligned}
\tag{3a} \tag{3b} \tag{3c}$$

Note the special case for vertex v (inequality 4c). By denoting $\beta_i = \lambda_i - \lambda'_i$, one can simplify the dual LP:

$$\begin{aligned}
\max \quad & \sum_{i \in T_v} a_i \beta_i \\
\text{subject to} \quad & -1 \leq \beta_i \leq 1 \quad \forall i \in T_v \quad (5a) \\
& \beta_i + \alpha_i - \alpha_{A(i)} \leq 0 \quad \forall i \in T_v - v \quad (5b) \\
& \beta_v + \alpha_v \leq 0 \quad (5c) \\
& \alpha_i \geq 0 \quad \forall i \in T_v \quad (5d)
\end{aligned}$$

We now summarize the necessary complementary slackness conditions required by the dual:

$$x_i > a_i \Rightarrow \lambda_i = 0, \lambda'_i = 1 \quad (\beta_i = -1) \tag{C1}$$

$$x_i < a_i \Rightarrow \lambda_i = 1, \lambda'_i = 0 \quad (\beta_i = 1) \tag{C2}$$

$$x_i > \sum_{j \in C(i)} x_j \Rightarrow \alpha_i = 0 \tag{C3}$$

$$x_i > 0 \Rightarrow \lambda_i - \lambda'_i + \alpha_i - \alpha_{p(i)} = 0 \tag{C4}$$

Since throughout the execution of the while loop $x_v \geq a_v$ and the case where $x_v = a_v$ is trivial, we will assume from now on that $x_v > a_v$. This implies the last necessary condition:

$$x_v > a_v \Rightarrow \alpha_v = 1 \tag{C5}$$

We begin by suggesting an initial assignment which might not be feasible, and in addition, might violate one of the complementary slackness properties.

The following lemma is a direct consequence of the construction of the dual LP and the complementary slackness constraints. It refers to a family of assignments to the dual LP that satisfy a *subset* of the complementary slackness conditions.

Lemma 2. *Let \mathbf{x}, \mathbf{d} be a feasible solution for the primal such that the sub-trees rooted in v are optimally assigned and v admits no Push-Path improvements. Let α, β be an assignment for the dual variables such that the following holds:*

$$\begin{cases} \alpha_i = \alpha_{p(i)} - \beta_i, & \text{if } x_i > 0 \\ \alpha_i \leq \alpha_{p(i)} - \beta_i, & \text{otherwise} \end{cases} \quad \beta_i = \begin{cases} -1, & \text{if } x_i > a_i \\ 1, & \text{if } x_i < a_i \\ \text{a value in } [-1, 1], & \text{if } x_i = a_i \end{cases} \quad (6)$$

*Then α, β satisfy all the properties of a feasible dual solution, and (α, β) along with (\mathbf{x}, \mathbf{d}) satisfy complementary slackness **except** that α_i might be negative for some nodes, and condition C3 could be falsified.*

Next, we observe that if our modified dual LP admits an optimal *feasible* solution, then our range of possible values for α, β can be narrowed due the total unimodularity of the simplified constraint matrix of the dual LP:

Observation 3. *If the dual LP has an optimal and feasible solution, then it has an integral, feasible and optimal solution, as well. In particular, for every $i \in T_v$, $\beta_i \in \{-1, 0, 1\}$.*

Observation 3 can be verified by induction on the constraint matrix of the dual LP, in order to show that every square sub-matrix of it has a determinant of ± 1 .

The following lemma complements Lemma 2 by suggesting a concrete assignment for each β_i in the case whenever $x_i = a_i$.

Lemma 3. *Consider an assignment as described in Lemma 2. If we set $\beta_i = 1$ whenever $x_i = a_i$, then:*

$$\forall j \in T_v, x_j > \sum_{k: \text{child of } j} x_k \Rightarrow \alpha_j \leq 0$$

Proof. We prove the claim by way of contradiction. Suppose that the claim is false, and let j be the highest node for which the claim does not hold. That is, $x_j > \sum_{k \in C(j)} x_k$ and $\alpha_j > 0$. Consider $P_{j \rightarrow v}$, the path from j to v . As we are trying to prove an upper bound for α_j , we will assume that for every node k on the path from v to j $\alpha_k = \alpha_{p(k)} - \beta_k$, as lower values will only strengthen our claim. This implies

$$\alpha_j = - \sum_{k \in P_{j \rightarrow v}} \beta_k. \quad (7)$$

Since $\beta_i = 1$ for all nodes i such that $x_i \leq a_i$, and $\beta_i = -1$ otherwise, $\alpha_j > 0$ implies:

$$|\{k \in P_{j \rightarrow v} : x_i > a_i\}| > |\{k \in P_{j \rightarrow v} : x_i \leq a_i\}| \quad (8)$$

This implies that we can reduce all x values of nodes $P_{j \rightarrow v}$ by an amount of at most $x_j - \sum_{k \in C(j)} x_k$ so as to get a feasible solution with a better objective function value. However, this is exactly a push operation, thereby contradicting the assumption of no further path-paths. \square

The following corollary is the contra-positive statement of Lemma 3

Corollary 1. *If there exists a node $j \in T_v$ such that $\alpha_j > 0$ and $x_j - \sum_{k \in C(j)} x_k > 0$, then there exists an ancestor i of j such that $\beta_i \in \{0, -1\}$ and $x_i = a_i$.*

We now prove the main theorem by way of induction. We inductively assume that the sub-trees rooted in v have both an optimal setting for the primal LP, and there exists an integral and feasible solution for the dual LP that satisfy the complementary slackness conditions. Without loss of generality, we assume that no child i of v has $x_i = 0$, since otherwise, we could use its assignments without any modifications, as x_i does not harden the feasibility constraints of v .

Consider the assumed set of assignments for the sub-trees rooted in v . By the assumption, they have corresponding assignments to the dual LPs. Observe that since the conditions listed in Lemma 2 are a subset of the complementary slackness conditions, Lemma 2 applies to them automatically.

We will start from a tentative solution to the dual by initially set the (α, β) according to the assumed assignments, and set $\alpha_v = 1$, $\beta_v = -1$. We let \mathbf{s}_1 denote the above assignment. Notice that for each child i of v , the dual LP that corresponds to the current assignment had

$$\alpha_i + \beta_i = 0,$$

as i was the root (this is a strict equality as by our assumption $x_i > 0$). However, in the current LP, the corresponding dual inequality becomes

$$\beta_i + \alpha_i - \alpha_v = 0,$$

As $\alpha_v = 1$, this equality is therefore violated. In order to rectify this, we first raise all the α value (except v 's) by 1, and denote the resulting solution by \mathbf{s}_2 . Note that by the feasibility of the original assignments to the sub-trees and by the definition of \mathbf{s}_2 all the nodes in T_v have non-negative α values. Also observe that \mathbf{s}_2 now has all the properties listed in Lemma 2. Thus, by Lemma 2, we can conclude that \mathbf{s}_2 is a feasible solution to the dual LP, and \mathbf{s}_2 along with (\mathbf{x}, \mathbf{d}) satisfy complementary slackness except that complementary slackness condition C3 might be violated.

Our next step would be to adjust \mathbf{s}_2 so as to fix any violation condition of condition C3. Let W be the set of all infeasible nodes:

$$W = \{j \in T_v : x_j > \sum_{k \in C(j)} x_k \text{ and } \alpha_j > 0\} \quad (9)$$

By Corollary 1, for each $j \in W$ there exists an ancestor i such that (1) $x_i = a_i$ and (2) $\beta_i \in \{-1, 0\}$. We let

$$X = \{i \in T_v : x_i = a_i, \beta_i \in \{-1, 0\}\} \quad (10)$$

Moreover, we let

$$Y = \{i \in X : \text{there is no ancestor of } i \text{ in } X\} \quad (11)$$

Thus, for each node $j \in W$ there exists an ancestor $i \in Y$.

We now define the final solution to the dual LP. Define assignment \mathbf{s}_3 to the dual LP for T_v by taking solution \mathbf{s}_2 with the following modifications:

1. $\forall k \in T_i$, such that $i \in Y$, subtract α_k by 1.

2. $\forall i \in Y$ add 1 to β_j .

Increasing the β_j values by 1 makes sure that complementary slackness condition C4 is satisfied after applying the first step. Applying the first modification step guarantees that complementary slackness condition C3 is again satisfied, as all nodes in W undergo the first modification. Observe that by definition, all the sub-trees rooted in nodes in Y are pair-wise disjoint. Hence, each α value can be decremented at most once. Also observe that in addition to nodes in W , other nodes may have their α values decremented. However, as by the definition of W , these nodes do not need to maintain condition C3, and thus this step will not violate their constraints. In addition, their α values are guaranteed to remain non-negative as they were previously incremented by 1.

In conclusion, all of the complementary slackness conditions for the dual LP now hold for (\mathbf{x}, \mathbf{d}) and \mathbf{s}_3 . Therefore, (\mathbf{x}, \mathbf{d}) is an optimal solution for T_v . \square

5 The Algorithm

Given the general technique presented in Algorithm 1, we conclude our results by giving an $O(n^2)$ algorithm that follows the spirit of improving by pushing the surplus from a given vertex downward, along a path. Recall that the algorithm *Push-Improve* performs the push operations *one* path at a time. Instead, we can leverage the fact that some paths can share the same prefix. Specifically, instead of the inner while-loop, executed for each node v in the tree, we introduce a depth-first-search algorithm in which for each node $j \in T_v$, the algorithm remembers the maximal amount, pushable through $P_{v \rightarrow j}$.

We make use of two measures, defined for each node $u \in T_v$. Let $\delta_u = |\{j \in P_{v \rightarrow u} : x_j > a_j\}| - |\{j \in P_{v \rightarrow u} : x_j \leq a_j\}|$. In other words, for any push operation along $P_{v \rightarrow u}$, δ_u is the difference between the number of nodes that will improve the objective function value, and the number of nodes that will worsen the objective function value, if we push a small enough value through $P_{v \rightarrow u}$. Additionally, we define the positive bottleneck along $P_{v \rightarrow u}$ as $\epsilon_u = \min_{j \in P_{v \rightarrow u}} \{x_j - a_j : x_j > a_j\}$. This is the maximum ϵ we can push on the path $P_{v \rightarrow u}$ while gaining exactly $\delta_u \epsilon$ in the objective function. In order to maintain feasibility, we restrict ϵ_u to be *no more* than x_k , for any node k on $P_{v \rightarrow u}$. This value will have a similar function as the ϵ value given in Algorithm 1. That is, for the current node v , and a successor u , ϵ_u will serve as the amount of *excess* we push through $P_{v \rightarrow u}$. Our algorithm will maintain feasibility by restricting the decrease in x_u by the sum of the decreases made on its direct children of u (unless x_u was strictly bigger than the sum of the x 's of its children before the decrease.)

The final algorithm for optimizing the assignment to T_v , can be seen as Algorithm 4 in Appendix E. It differs from Algorithm 1 in the way the sub-tree T_v is modified for each node v .

The following theorem states that Algorithm 4 is optimal.

Theorem 5.1. *When Algorithm 4 leaves node $v \in V$, there is no push-path going from the root r , ends at a leaf, and passes through v .*

Proof. First, we note the following observation, which suggests that the potential for improvement on any path from the root to a node v cannot increase.

Observation 4. *Let u be a node in T . Let*

$$\delta_u^* = |\{i \in P_{r \rightarrow u} : x_v > a_v\}| - |\{i \in P_{r \rightarrow u} : x_v \leq a_v\}|.$$

δ_u^ does not increase throughout the execution of the algorithm.*

The observation follows immediately from the fact that the only modifications to the x values of the nodes are decreases.

We proceed to prove the lemma by induction on the height h of the node v . For $h = 0$ (leaves), the claim is trivial. Assume the claim holds for $h = k$, and let v be a node of height $k + 1$. The claim follows immediately from Observation 4: no sub-tree rooted in a child of v can be improved as a result of a push-path through it. Additionally, the path from r to v never becomes amenable to improvements through push operations, once the algorithm leaves v . This concludes the proof. \square

Running Time The algorithm essentially performs a depth-first-search for every node v on the tree. Therefore, the running time of the algorithm is $O(n^2)$.

6 Conclusions and Future Work

We have demonstrated the technical difficulties that our problem entails, as well as an efficient method for handling a broad class of instances of the problem. Due to their high efficiency, our methods can be run on relatively large instances in practice. We also believe that our algorithm might be applicable to settings beyond recommendation systems.

An immediate open question is to extend our algorithm to the case of general DAGs. It seems that one needs some new ideas to give a combinatorial algorithm for this general case. In fact even a (fast) approximation algorithm for this case seems to be beyond the reach of our techniques. Another interesting direction would be to consider other measures such as the ℓ_2 -norm. Due to the fundamental difference between the ℓ_1 and ℓ_2 norms, we suspect that this different distance measure will require a completely different approach.

In addition to considering alternative objective functions, we can also consider other constraints. For instance, we can consider comparing the value assigned to each node to the *average* value of its children. Another type of constraint would be to require equality between the value of a node, and the sum of the values of its children.

References

- [AHKW06] Stanislav Angelov, Boulos Harb, Sampath Kannan, and Li-San Wang. Weighted isotonic regression under the ℓ_1 norm. In *SODA*, pages 783–791, 2006.
- [BBBB72] R. E. Barlow, D. J. Bartholomew, J. M. Bremner, and H. D. Brunk. *Statistical Inference Under Order Restrictions*. Wiley, 1972.
- [BC90] M. J. Best and N. Chakravarti. Active set algorithms for isotonic regression: a unifying framework. *Math. Program.*, 47:425–439, August 1990.
- [CKP07] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. Page-level template detection via isotonic smoothing. In *WWW*, pages 61–70, 2007.
- [DCZ⁺10] Anlei Dong, Yi Chang, Zhaohui Zheng, Gilad Mishne, Jing Bai, Ruiqiang Zhang, Karolina Buchner, Ciya Liao, and Fernando Diaz. Towards recency ranking in web search. In *WSDM*, pages 11–20, 2010.
- [DKK11] Gideon Dror, Noam Koenigstein, and Yehuda Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *Recommender Systems*, Chicago, IL, USA, 2011. ACM.

- [Fei98] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45:634–652, July 1998.
- [Fle04] Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 1001–1010, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
- [FT84] J. Friedman and R. Tibshirani. The monotone smoothing of scatterplots. *Technometrics*, 1984.
- [KFB09] Rémon Kamp, Ad Feelders, and Nicola Barile. Isotonic classification trees. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, IDA '09, pages 405–416, Berlin, Heidelberg, 2009. Springer-Verlag.
- [LB01] Klervi Leuraud and Jacques Benichou. A comparison of several methods to test for the existence of a monotonic dose-response relationship in clinical and epidemiological studies. *Statistics in Medicine*, 20(22):3335–3351, 2001.
- [MAC01] Jessica Y. Mancuso, Hongshik Ahn, and James J. Chen. Order-restricted dose-related trend tests. *Statistics in Medicine*, 20(15):2305–2318, 2001.
- [MJDP⁺00] Tony Morton-Jones, Peter Diggle, Louise Parker, Heather O. Dickinson, and Keith Binks. Additive isotonic regression models in epidemiology. *Statistics in Medicine*, 19(6):849–859, 2000.
- [MSCZ10] Taesup Moon, Alex J. Smola, Yi Chang, and Zhaohui Zheng. Intervalrank: isotonic regression with listwise and pairwise constraints. In *WSDM*, pages 151–160, 2010.
- [PG08] Kunal Punera and Joydeep Ghosh. Enhanced hierarchical classification via isotonic smoothing. In *WWW*, pages 151–160, 2008.
- [RWD88] T. ROBERTSON, F. T. Wright, and R. L. Dykstra. *Order Restricted Statistical Inference*. Wiley, 1988.
- [Sto] Quentin F. Stout. Isotonic Regression Algorithms. <http://www.eecs.umich.edu/~qstout/IsoRegAlg.html>. Retrieved: Aug. 6th, 2011.
- [Sto08] Quentin F. Stout. Unimodal regression via prefix isotonic regression. *Computational Statistics & Data Analysis*, 53(2):289–297, 2008.
- [Ulm86] K. Ulm. Nonparametric analysis of dose-response relations in epidemiology. *Mathematical Modelling*, 7(5-8):777 – 783, 1986.

A The Weighted ℓ_1 -Norm Case

We now discuss the case where the nodes on the given tree can have varying levels of importance with respect to the objective function. Specifically, as done in related studies, we consider the case in which for each node $i \in V$, there is an associated weight w_i . For ease of presentation, we assume that all weights are integral, i.e. $\mathbf{w} \in \mathbb{N}_{\geq}^n$. The objective function will be $g(\mathbf{x}) = \sum_{i \in V} w_i \cdot |a_i - x_i|$.

Hence, we can simply reinterpret the variable δ_i defined for Algorithm 4 as the weighted balance between the nodes which will benefit, and the nodes that will “suffer” as a result of a Push-Path operation. More precisely, when considering the path $P_{v \rightarrow i}$, we will compute $\delta_i = \sum_{j \in P_{v \rightarrow i}: x_j > a_j} w_j - \sum_{j \in P_{v \rightarrow i}: x_j \leq a_j} w_j$. Therefore, a feasible push-improvement across a path $P_{v \rightarrow u}$ would lead to an improvement if and only if $\delta_u > 0$.

The above discussion leads to the following simple modification to procedure **SetParams**, given in Algorithm 2. Note that the weighted case reduces to the unweighted case by setting the weights

Algorithm 2: The modified **Set-Params** procedure for the weighted case

Input: Vertex v

```

1 Set-Params(Vertex  $i$ , Non-negative integer  $\delta$ , Non-negative real-value  $\epsilon$ )
2 begin
3   if  $x_i > a_i$  then
4      $\delta_i \leftarrow \delta + w_i, \epsilon_i \leftarrow \min\{\epsilon, x_i - a_i\}$ 
5   end
6   else
7      $\delta_i \leftarrow \delta - w_i, \epsilon_i \leftarrow \min\{x_i, \epsilon\}$ 
8   end
9 end

```

to 1. Clearly, the algorithm has the same $O(n^2)$ running time of the original algorithm. The following theorem argues about the optimality of the modified algorithm:

Theorem A.1. *The algorithm resulting from the modification given in Algorithm 2 obtains the optimal weighted- ℓ_1 objective function value.*

Proof. In order to argue about the correctness of the modified algorithm, we compare the objective function obtained by the algorithm to the one obtained by the original algorithm, on an equivalent unweighted tree.

The construction We construct the tree $\tilde{T} = (\tilde{V}, \tilde{E})$ by replacing each node i with a chain i_1, \dots, i_{w_i} , such that for any $1 \leq j < w_i$, $(i_j, i_{j+1}) \in \tilde{E}$. Additionally, we set for children $k \in C(i)$ $(k_{w_k}, i_1) \in \tilde{E}$, and for i 's parent ℓ $(i_{w_i}, \ell_1) \in \tilde{E}$.

It is easy to see that \tilde{T} is a tree. Notice that \tilde{T} might be arbitrarily large (according to the weights). However, it is used only for the sake of proof of correctness, and never actually constructed by the algorithm. The following proof sketch highlights the equivalence of the uniform weight case to the weighted case.

Claim 2. *Let \mathbf{x} and $\tilde{\mathbf{x}}$ be optimal assignments for T and \tilde{T} , respectively. Then $g(\mathbf{x}) = f(\tilde{\mathbf{x}})$.*

The following immediate observation, which follows from the construction of \tilde{T} , implies the above claim.

Observation 5. *Let $\tilde{\mathbf{x}}$ be an optimal assignment for \tilde{T} . Then for any chain (i_1, \dots, i_{w_i}) that corresponds to vertex i in T :*

$$x_{i_1} = x_{i_2} = \dots = x_{i_{w_i}}$$

The following claim complements claim 2:

Claim 3. Let \mathbf{x} and $\tilde{\mathbf{x}}$ be the feasible assignments returned by Algorithm 4 and the modified algorithm for weighted trees, respectively. Then

$$g(\mathbf{x}) = f(\tilde{\mathbf{x}})$$

□

B The ℓ_∞ -norm

We now turn our attention to the case of the ℓ_∞ -norm; i.e. minimizing the maximal difference $\max_{u \in V} |a_i - x_i|$. In contrast to the case of the ℓ_1 -norm, this optimization problem can be solved in a straightforward manner by using dynamic programming, even when the underlying graph is a directed acyclic graph.

For a given value $t \geq 0$, the algorithm will go over all nodes and tries to produce an assignment of objective value at most t . We can show that if the algorithm fails then there is no valid assignment of objective value at most t . To find the optimal objective value then one only needs to run a binary search on the variable t .

Algorithm 3: The dynamic programming algorithm for the ℓ_∞ -norm case

Input: DAG $G=(V,E)$, vertices $1, \dots, n$ sorted in topological order, vertex weight vector \mathbf{a} .
1 for $i \leftarrow 1$ **to** n **do**
2 | $x_i^{min} \leftarrow \max\{0, \sum_{j \in C(i)} x_j^{min}, a_i - t\}$
3 end
4 return \mathbf{x}^{min}

As mentioned above, we perform a binary search on t in the range $[0, \sum_i a_i]$. Clearly, for an instance of the problem with optimal solution value τ , the running time of Algorithm 3 would be $O(n \cdot \log \tau)$. We now briefly outline the proof of correctness of the algorithm.

Theorem B.1. For any given $t \geq 0$, $\mathbf{x} = \mathbf{x}^{min}$ is a valid solution. Furthermore, if $\|\mathbf{x} - \mathbf{a}\|_\infty > t$, then there does not exist a valid solution \mathbf{x}' such that $\|\mathbf{a} - \mathbf{x}'\|_\infty \leq t$.

Proof. The validity of \mathbf{x} follows from definition. To prove the second part we show the following simple lemma.

Lemma 4. If \mathbf{x}' is a valid solution and $\|\mathbf{a} - \mathbf{x}'\|_\infty \leq t$, then for all i , $x'_i \geq x_i^{min}$

Proof. The proof follows with a simple induction on i . Note that because $\|\mathbf{a} - \mathbf{x}'\|_\infty \leq t$, $x'_i \geq a_i - t$. Furthermore, \mathbf{x}' is a valid solution so $x'_i \geq 0$ and

$$x'_i \geq \sum_{j \in C(i)} x'_j \geq \sum_{j \in C(i)} x_j^{min} = x_i^{min},$$

where we have used the induction hypothesis for the second inequality. It then follows that,

$$x'_i \geq \max\{0, \sum_{j \in C(i)} x_j^{min}, a_i - t\} = x_i^{min}. \quad \square$$

Now assume that there is a valid solution \mathbf{x}' with objective value at most t . It follows that for all i ,

$$a_i - t \leq x_i^{\min} \leq x'_i \leq a_i + t,$$

that is, $\|\mathbf{x}^{\min} - \mathbf{a}\|_\infty \leq t$. □

C Hardness of Approximation in General Graphs

As mentioned before when the objective value is the ℓ_1 norm of the difference between the \mathbf{x} and \mathbf{a} vectors the (most general case of the) problem can be solved exactly in polynomial time by solving a linear program. In fact, it is not hard to see that using a similar approach one can solve this general case of the problem for *any* ℓ_p norm with $1 \leq p \leq \infty$. The only difference is that one has a linear program with infinitely many facets which has an efficient separation oracle and can be solved with the Ellipsoid method.

For an instance where all the input values are integral, one might ask whether the task of finding an optimal *integral* solution is tractable or not. This is especially interesting for the ℓ_1 case, since in the case of trees, an integral solution can be found efficiently by our algorithm of Section 4, if the initial a values are integrals. Unfortunately, as soon as one considers the DAG case (even the special case of layered dags) this problem becomes intractable for essentially all ℓ_p norms. The following theorem summarizes our hardness results.

Theorem C.1. *Unless $NP \subseteq TIME(n^{O(\log \log n)})$ it is NP-hard to approximate the Integral Isotonic Regression problem for the case of directed acyclic graphs better than $\Theta((\log n)^{1/p})$ for the ℓ_p norm.*

Proof. We prove the theorem by a reduction from the *Set Cover* problem. In the Set Cover problem one is given sets S_1, S_2, \dots, S_m such that $S_1 \cup S_2 \cup \dots \cup S_m = \{1, 2, \dots, n\}$ and the objective is to select a minimum number of S_i 's such that their union is still $\{1, 2, \dots, n\}$. It is a well known result of Feige [Fei98] that unless $NP \subseteq TIME(n^{O(\log \log n)})$ it is NP-hard to approximate Set Cover better than a factor of $(1 - o(1)) \log n$. Our reduction uses vertex weights so as to simplify the construction. However, one can easily adapt the construction to the uniform case by adding multiple copies of nodes so as to simulate large weights.

Given an instance of the Set Cover problem We construct the following instance of the SBHSP problem:

- The vertex set of the output digraph will be $V = \{v_1, \dots, v_m, u_1, \dots, u_n\}$.
- The edge set of the output digraph will be $E = \{(v_i, u_j) : j \in S_i\}$.
- The a values on the vertices will be as follows. For all v_i we have $a(v_i) = 1$, while for all u_j we have $a(u_j) = |\{i : j \in S_i\}| - 1$.
- The w values (weights) of the vertices will be as follows. For all v_i we have $w(v_i) = 1$, while for all u_j we have $w(u_j) = m$.

On the one hand it is easy to see that for any set cover of the original instance (of size α) one can construct a solution to the SBHSP instance (of cost $\sqrt[p]{\alpha}$) by assigning $x(v_i) = 0$ if S_i is selected and 1 otherwise, and $x(u_j) = a(u_j)$ for all u_j .

On the other hand it is not hard to see that the optimal solution to the SBHSP will have $x(v_i) \in \{0, 1\}$ for all i and $x(u_j) = a(u_j)$ for all j . Furthermore, for any such solution (of cost

α) the set of S_i for which $x(v_i) = 0$ can be easily seen to be a valid set cover of size α^p . Hence, a hardness of approximation of $(1 - o(1)) \log n$ for the Set Cover problem implies a hardness of approximation of $\Theta(\sqrt[p]{\log n})$ for SBHSP when the objective value is defined using the ℓ_p norm for any $1 \leq p < \infty$. \square

Remark 1. The hard-instances of Set-Cover generated by Feige [Fei98] have less Sets than elements. As a result it is not hard to see that the hardness achieved by the proof of the above theorem is in fact $((1 - o(1)) \ln n)^{1/p}$ for the weighted case and $\left(\frac{(1-o(1)) \ln n}{2}\right)^{1/p}$ for the non-weighted case.

D FPTAS for optimizing under the ℓ_1 norm for Bilayered graphs

Consider a DAG $G = (V, E)$ which is bilayered, i.e. the vertex set can be partitioned as $V = U \cup W$ and each edge is from the U side to the W side ($E \subseteq U \times W$.) In this section we show a fast Fully Polynomial Approximation Scheme for SBHSP with the ℓ_1 norm for such DAGs. The run time will be close to linear in the size of the DAG. The algorithm is a simple reduction to a well known class of problems which admit such FPTASes. These problems are restricted class of the Mixed Positive Packing and Covering Problem, see [Fle04].

We start by the following simple observation.

Lemma 5. *When optimizing the ℓ_1 norm and when the input DAG is bilayered there is always an optimal solution with the following two properties, (i) $\forall w \in W : x_w = a_w$, (ii) $\forall u \in U : x_u \leq a_u$.*

Proof. Consider any optimal solution \mathbf{x} , and a vertex $w \in W$ for which $x_w \neq a_w$. If $x_w < a_w$ changing the assigned value of this vertex to a_w produces another valid solution with a better objective function value. Now consider the case in which $x_w > a_w$. If $x_w > \sum_{u \in C(w)} x_u$ then again we can improve the objective function by decreasing x_w slightly, and if $x_w = \sum_{u \in C(w)} x_u$ we can simultaneously decrease x_w and the assigned value of some of its children. This last step would help the objective value due to the improvement on w , and possibly hurt it by the exact same amount due to the decrease on its children, while maintaining a valid the solution. Doing this step on every node on the W side results in a solution that satisfies the first condition.

For the second condition observe that if $x_u > a_u$ we can simply decrease it to a_u without changing the validity of the solution while decreasing the objective function. In other words any optimal solution must satisfy the second condition. \square

Given the above lemma one can write the following linear program whose solution is the exact value of the optimal solution. The left hand side is the original program based on the LP (3a)-(3d) while the right hand side is the result of a simplification.

$$\begin{array}{ll}
\min & \sum_{v \in V} d_v \\
\text{subject to} & d_u \geq 0 \quad \forall u \in U \\
& x_u \geq 0 \quad \forall u \in U \\
& d_u + x_u = a_u \quad \forall u \in U \\
& \sum_{u \in C(w)} x_u \leq a_w \quad \forall w \in W
\end{array}
\qquad
\begin{array}{ll}
\min & \sum_{v \in V} d_v \\
\text{subject to} & d_u \geq 0 \quad \forall u \in U \quad (13a) \\
& d_u \leq a_u \quad \forall u \in U \quad (13b) \\
& \sum_{u \in C(w)} d_u \geq \left(\sum_{u \in C(w)} a_u \right) - a_w \quad \forall w \in W \quad (13c)
\end{array}$$

Once written in this form the above formulation is a, so called, Mixed Positive Packing and Covering

Program. In fact it is among a certain class of such programs for which Fleischer [Fle04] provides a fast FPTAS. In particular, we have the following theorem.

Theorem D.1. *When the input is a bilayered graph and the objective value is in terms of the ℓ_1 norm, there is an algorithm that given $\epsilon > 0$ runs in time $O(|V||E| \log(|V|)/\epsilon^2)$ and returns a valid solution with objective value no more than $(1 + \epsilon)$ times that of the optimum.*

Proof. The proof is a simple application of Theorem 2.1 from [Fle04] to the above Linear Program. A simple corollary of that theorem is that the algorithm finishes in $O(|V||E| \log(|V|)/\epsilon^2)$ steps¹ and in each step one has to find the most unsatisfied constraint among (13a)-(13c) given a current solution $\bar{\mathbf{d}}$. Each such step can be done by evaluating all the constraints in total time $|E|$. \square

E Omitted figures and algorithms

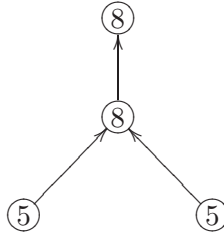


Figure 1: A counter-example for the naive algorithm. Node annotations denote the a values. The naive algorithm will obtain an objective function value of 4, whereas the optimum value is 2.

¹the constant C in Theorem 2.1 of [Fle04] is 1 in our case.

Algorithm 4: The improved ImproveSubtree procedure

Input: Vertex v

```
1 begin
  | /* If  $x_v = a_v$   $T(v)$  is optimal */
2   if  $x_v > a_v$  then
3   |   Push-Search( $v, \infty, 0$ )
4   end
5 end

6 Push-Search(Vertex  $u$ , Non-negative real-value  $\epsilon$ , Non-negative Integer  $\delta$ )
7 begin
8   Set-Params( $u, \epsilon, \delta$ )
9   if  $\epsilon_u = 0$  then
10  |   return 0
11  end
12   $sum \leftarrow 0$ 
13   $\ell \leftarrow \min\{\epsilon_u, x_u - \sum_{k:\text{child of } u} x_k\}$ 
14  if  $\ell > 0$  and  $\delta_u > 0$  then
15  |    $x_u \leftarrow x_u - \ell, sum \leftarrow \ell$ 
16  |   Set-Params( $u, \delta, \epsilon - \ell$ )
17  end
18  foreach  $j \in c(u)$  do
19  |   if  $sum = \epsilon_u$  then
20  | |   return 0 /* Speedup
21  |   end
22  |    $t \leftarrow \text{Push-Search}(j, \epsilon_u, \delta_u)$ 
23  |    $sum \leftarrow sum + t, x_u \leftarrow x_u - t$ 
24  |   Set-Params( $u, \delta, \epsilon - sum$ )
25  end
26  return  $sum$ 
27 end

28 Set-Params(Vertex  $i$ , Non-negative integer  $\delta$ , Non-negative real-value  $\epsilon$ )
29 if  $x_i > a_i$  then
30 |    $\delta_i \leftarrow \delta + 1, \epsilon_i \leftarrow \min\{\epsilon, x_i - a_i\}$ 
31 end
32 else
33 |    $\delta_i \leftarrow \delta - 1, \epsilon_i \leftarrow \min\{x_i, \epsilon\}$ 
34 end
```
